



Objektorientierung und komplexe Datentypen

Was bedeutet objekt-orientiert?

- Programmierung ist die Umsetzung von Sachverhalten und Aufgaben
- Zusammenfassen und Aufteilen von Sachverhalten
- einzelne Aspekte werden als Objekte abstrahiert
- diese Objekte haben eine Identität, einen Zustand und zeigen ein bestimmtes Verhalten

Klassen

- Klasse gibt die Struktur, Form, das Modell des Objektes vor
- enthält Attribute (Zustand) und Methoden (Verhalten)
- Definition von komplexen Datentypen
- Aufteilung von komplexen Programmen
- leichte Wiederverwendbarkeit

Objekte

- Klassenname als Bezeichner (Typname)
- Variable enthält nicht direkt Wert, sondern die Referenz auf das Objekt
- durch Konstruktoraufruf wird Objekt existent (Instanziierung, **new**)
- Garbage Collection räumt Objekte ohne Referenzen aus dem Speicher

```
Hase h1 = new Hase();           // h1.ohren == 2 (default)
Hase h2 = h1;                   // h1.ohren == 2, h2.ohren = 2
h1.ohren = 0;                     // h1.ohren == 0, h2.ohren = 0
```

Sichtbarkeit

- Attribute und Methoden können in Verwendung eingeschränkt werden
- Beispiel-Modifizierer
 - public freier Zugriff
 - <kein> Zugriff innerhalb des Paketes
 - private Zugriff nur innerhalb der Klasse
 - protected Zugriff nur innerhalb der Klasse und abgeleiteten Klassen
 - static gleicher Wert für alle Instanzen
 - final ein Initialzustand wird festgeschrieben

Attribute

- Variablen, die den Objektzustand speichern
- Objektattribute: `objektName.attributName`
- Klassenattribute: `KlassenName.attributName`
- **this** ist die Referenz auf das derzeitige Objekt

```
Hase welcherHaseBistdu() {  
    return this;  
}
```

Attribute

- Variablen, die den Objektzustand speichern
- Objektattribute: `objektName.attributName`
- Klassenattribute: `KlassenName.attributName`
- **this** ist die Referenz auf das derzeitige Objekt

```
int ohren = 2;
...
int ohrSumme(int ohren) {
    return ohren + this.ohren;
}
```

Methoden

- Operationen, die das Verhalten definieren
- Konstruktoren als spezielle Methoden besitzen keinen Rückgabetyt
- Objektmethoden: `objektName.methodenName(...)`
- Klassenmethoden: `KlassenName.methodenName(..)`
- statische Methoden können nicht direkt auf Objektvariablen zugreifen

Übergabe von Variablen: call-by-value

- primitive Datentypen

```
int i = 5;
```

```
int j = i;
```

```
System.out.println(i); // 5
```

```
i = 1;
```

```
System.out.println(j); // 5
```

Übergabe von Variablen: call-by-value

- komplexe Datentypen

```
public static void main ( String [] args ) {  
    Hase h = new Hase ();  
    h.ohren = 0;  
    Hase.ersetzeOhr(h);  
    // h.ohren == 1;  
}  
  
static void ersetzeOhr(Hase m) {  
    // m.ohren == 0;  
    m.ohren = 1;  
}
```

Übergabe von Variablen: call-by-value

- komplexe Datentypen

```
public static void main ( String [] args ) {  
    Hase h = new Hase ();  
    h.ohren = 0;  
    Hase.ersetzeHasen(h);  
    // h.ohren == 0;  
}  
  
static void ersetzeHasen(Hase m) {  
    m = new Hase();  
}
```

Vergleiche von Variablen

- für primitive Datentypen

```
int i = 1;
```

```
int j = 2;
```

```
System.out.println( i == j );    // false
```

Vergleiche von Variablen

- für primitive Datentypen

```
int i = 1;
int j = 2;
System.out.println( i == j );    // false
```

- für komplexe Datentypen

- „==“ überprüft auf Identität
- equals() überprüft Gleichheit

```
String i = new String("Progwerkstatt");
System.out.println(i);           // Progwerkstatt
System.out.println( i == "Progwerkstatt" );    // false
System.out.println( i.equals("Progwerkstatt") ); // true
```

Aufgabe

- Klasse zur Datenrepräsentation von Vektoren ($n = 3$)
- Attribute, Methoden, Konstruktoren deklarieren
- Methode `length()` um die Länge des Vektors zu berechnen
- Methode `addiere(Vektor v)`, die einen Vektor um einen anderen erweitert
- Klassenmethode `addition(Vektor v1, Vektor v2)`, die zwei Vektoren addiert und einen Neuen zurückgibt
- Testen und Ausgabe von Ergebnissen